Using Coven to Profile and Tune Parallel Programs

Nathan A. DeBardeleben
864-656-5909
ndebard@parl.clemson.edu

Vishal Patil
864-656-5909
vpatil@parl.clemson.edu

Walter B. Ligon III
864-656-1224
walt@parl.clemson.edu

Parallel Architecture Research Lab
Department of Engineering
Clemson University
105 Riggs Hall
Clemson, SC 29634-0915
http://www.parl.clemson.edu

Abstract

Coven is a framework for creating Problem Solving Environments (PSEs) for parallel computers. Programs are composed using a special language for describing the flow of data between modules. Modules are supplied by the user and written in C or Fortran. Coven provides a runtime environment complete with multithreading, shared memory utilization between threads, and automatic profiling of trace information. It has been used to create PSEs for satellite remote sensing, molecular dynamics, n-body simulations, and complex fluid dynamics.

Identification and tuning of performance problems in parallel applications can be difficult. PSEs for parallel computers that assist the application domain scientist in developing parallel codes must also provide a facility to help tune these codes. This work presents a case study of a performance analysis and profiling system for Coven. We show how several parallel programs with poor performance can be optimized with the help of the Coven profiler. Additionally, we provide a detailed description of Coven's profiling system.

1. Introduction

Problem Solving Environments (PSEs) are becoming an integral part of of modern high performance computing (HPC) due to the increasing complexity of the types of simulations being run and the underlying problems being modeled, as well as the increasing complexity of the computer systems being employed. PSEs assist users

in developing complex codes in many ways. They often are targeted at a particular application domain, and have pre-built tools and ready to use pieces of application code. These properties help encourage code reuse and quicker development time, especially during the design and test cycle. However, by abstracting away some of the complexities involved in developing scientific application codes, performance problems can be hidden and hard to detect, especially in parallel applications. Thus, a problem solving environment needs to provide performance analysis and profiling facilities in order to assist users in tuning their parallel applications for improved performance.

In previous work[6, 7] we have presented the Coven framework for developing HPC problem solving environments. Coven brings together an extensible and reusable component programming model, a powerful runtime engine, and a suite of tools to ease application development, debugging, and performance evaluation and tuning. In this paper, we present a case study of the performance analysis and profiling system for Coven.

Some related PSEs for high performance computers are BioPSE and Uintah[4] (based on SCIRun[10]), and CERSe[7] (based on Coven[6]). Uintah targets distributed memory machines and uses the Common Component Architecture[3] component model. The Common Component Architecture[3], SCIRun and Coven are problem solving environment toolkits that can be extended to build additional environments.

The Multi-Processing Environment (MPE)[2] is a library for profiling MPI applications. MPE logs all MPI function calls, recording timing and trace event information. While MPE provides a way to analyze MPI events, the Coven profiling system extends MPE to profile Coven events as well. This allows the Coven profiler to present and analyze trace information of not only MPI events but also CPU load, memory usage, page faults, Coven modules, and Coven Threads. TAU[14] is a performance monitoring system which allows for instrumentation of code during compilation and execution to monitor trace and performance statistics of programs. Coven's trace files can be analyzed by any MPE log viewer. Similarly, TAU interfaces with other tools, such as VAMPIR[13], a commercial performance and trace visualizer. The Uintah environment also uses TAU, as well as XPARE[5]. XPARE allows for tracking of performance information during an application's development cycle. It can then be used to identify when a negative shift in the performance of the application is detected due to changes in the code. Paradyn[12] is another tool that analyzes parallel program performance by augmenting user code and using a search model to identify potential performance problems. Like Coven, Paradyn allows for visualization of profiling information as well as a way for programmers to add new visualizations.

In the next section we provide a brief overview of Coven and those features relevant to this paper. In subsequent sections we present the methodology behind Coven's back-end and front-end profiling system. We then present a

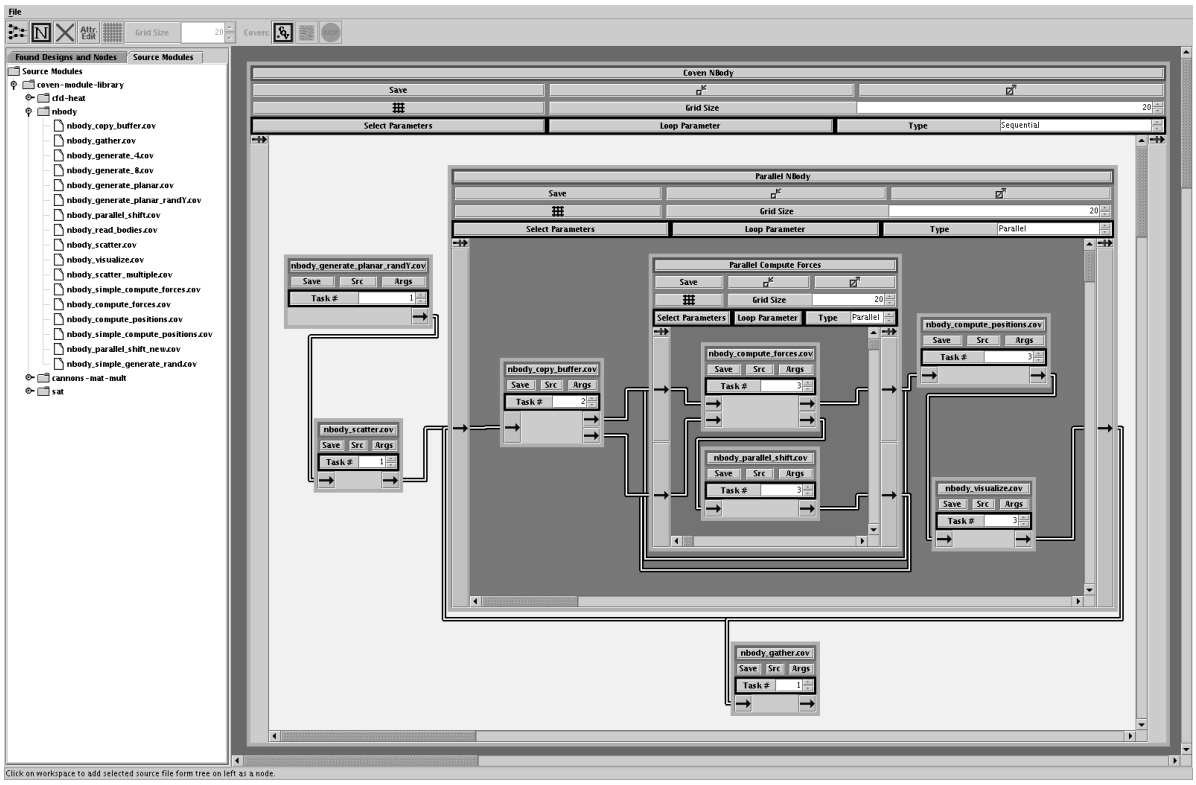

Figure 1. Graphical Editor Agent

case study of using the Coven profiler on several parallel applications to improve performance.

2. Coven

Coven is a framework for building problem solving environments (PSEs) for parallel computers. Programs are created in Coven by connecting pieces of code called *modules*. These modules are supplied by the user and written in either C or Fortran. Modules look very much like function calls or routines but contain Coven-specific directives. The directives are used to describe the interface to the module, such as what data it consumes and what data it creates.

Applications are created using a custom Coven language that specifies how data flows through modules. A Java based GUI (Figure 1) is provided to help in the creation of the program file. The Coven program file is then processed by a code generator which translates the text into an internal Coven binary format. The runtime engine, running on a parallel machine, then uses this program file with the user's modules to run the application. Figure 2 depicts the relationship between these components.

Data in Coven is encapsulated in an internal data structure called a Tagged Partition Handle (TPH). This struc-

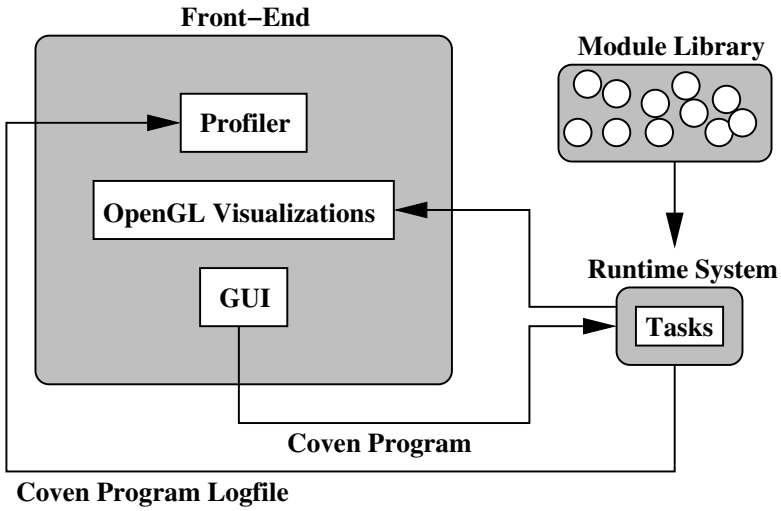

Figure 2. Coven Architecture

ture is completely transparent to the user, who can simply view their program as a collection of buffers flowing between modules.

Coven emulates multithreading by scheduling multiple MPI tasks on the same processor. These are called Coven Threads. We use MPICH[9], an MPI implementation that currently does not support multithreading. Therefore, we implement Coven Threads as processes scheduled on the same processor. Transparent inter-process shared memory is provided by Coven so that Coven Threads on the same processor can exchange data faster than by using MPI. MPICH2[11] provides a number of performance improvements which benefit Coven, particularly regarding multiple MPI tasks scheduled concurrently on the same processor.

Coven aims to provide a total solution for creating, running, debugging, profiling, and tuning parallel programs. Therefore, OpenGL dataset visualizations, graphical drag-and-drop program creation tools, and a program profiler are some of the tools which are provided.

Different types of PSEs have been built using Coven, including satellite remote sensing, molecular dynamics, complex fluid dynamics, and n-body simulations. Each of these environments contain custom graphical tools and reusable modules tailored to the specifically domain.

In the next section we discuss Coven's runtime engine profiling system and how it can automatically log program events for later analysis.

3. Runtime Engine Profiling

Coven's runtime engine defines a number of events which can occur during the lifetime of a Coven program. When these events occur, Coven automatically logs these events. With each event, the processor number, Coven Thread number, time, and additional information are saved. This allows for offline review of the steps which occurred during the Coven program run.

There are two classes of events namely: point events and rectangular events. Point events happen at an instance in time, like a snapshot. Rectangular events occur over a period of time. Many events are logged in Coven, including module execution, load a module placed on the CPU, current memory usage, length of TPH queues, TPHs being transferred, and others. Users can define new states through a simple interface and then log their own events as necessary.

The MPE[2] logging facility is used in Coven. MPE is a widely used profiling extension to MPI which works with any of its implementations. All MPI events are logged using MPE and the Coven runtime engine also defines many custom events. By using MPE, Coven program logfiles can be viewed by any MPE compatible profiler, such as Jumpshot, Nupshot, or Upshot. Additionally, MPE handles all issues regarding collecting profile logs from each processor and coalescing them into a single log file.

Each time an event is logged by MPE, additional user data can be appended. Coven uses this user data space to store information about an event, such as the current CPU load, TPH queue depth, or module number being executed.

To gather additional statistics, Coven consults the `/proc` file system on the node running the Coven Threads. Coven gathers information about the different Coven Threads running on a processor, such as the number of page faults, time scheduled in user mode, time scheduled in kernel mode, virtual memory size, and other statistics. These are then saved into the user data portion of the custom MPE events.

4. Front-End Profiling

Once a Coven program's runtime trace and profile information are logged, offline profiling can take place. The Coven profiler provides a graphical, Java based tool for analyzing these log files. Multiple visualizations for the

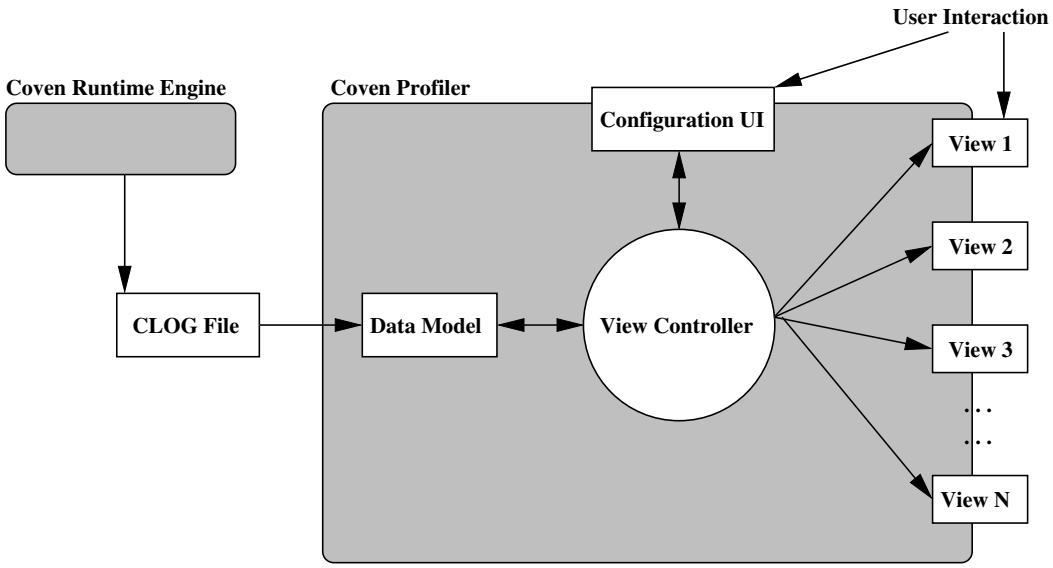

Figure 3. Profiler Architecture

various aspects of the parallel program execution are provided, thereby enabling the user to observe and analyze its behavior.

MPE supports three different logfile formats ALOG, CLOG and SLOG. ALOG is provided for backward compatibility purposes only and stores events as ASCII text. The default format is the CLOG format which stores a collection of events with single timestamps in a binary format. The SLOG format is also a binary format however it stores data as states. It is the most powerful format and enables visualization tools to handle logfiles containing gigabytes of data. Coven uses the CLOG instead of the more scalable SLOG because storage of the additional user data (which contains Coven specific information as well as information regarding the CPU load and memory) is currently facilitated only in the CLOG format.

The Coven profiler is composed of several components which interact with the log file to provide different visualizations to the user. Figure 3 depicts the relationship between these components, and each is described briefly below.

Data Model This component is responsible for parsing the input log files and initializing the various internal data structures which are used by the Views to graphically render the data. The Data Model uses a Jumpshot 4 API[1] to extract trace information from the various MPI events and collect the profile data for Coven-defined events.

Each Coven event contains additional information stored in the user data space. The data module correlates

the extracted data to obtain information regarding the Coven modules, auxiliary events, TPHs, memory consumption, and CPU load.

The Data Model is constructed by implementing a simple interface defined by the profiler. The profiler then uses this interface to extract information from the log file through the model. The advantage of this approach is that it is possible to use a another log format without affecting the rest of the system. This can be done by simply defining a new data model for the log format and implementomg the interface defined by the profiler.

View Controller This component is responsible for controlling the flow of data from the Data Model to the various visualizations. After the information is extracted from the log files and stored in the internal data structures, the View Controller uses them to initialize the various visualizations. Working in coordination with the Configuration UI, the View Controller is also responsible for reducing the information presented to the user as they refine the trace data they are interested in analyzing.

Configuration UI The Coven profiler provides a facility to refine the trace information that a user sees. This allows for the set of all events to be restricted to those of particular interest at one time. With this, a user can examine a subset of the processors, Coven Threads, modules, or auxiliary events. The Configuration UI interacts with the View Controller, communicating the user choices. The View Controller then filters out the information from the Data Model and updates the Views, thereby removing the unwanted information.

Views These form the visualizations that depict the different aspects of a parallel program run. The profiler also provides an interface to incorporate new visualizations. Creating new Views requires simply implementing this interface, upon which the View can interact with the Data Model API. Several visualizations are described next.

TPH Lifetime View (Figures 4 and 6) This visualization displays information regarding the flow of TPHs through various modules of the Coven program. Similar to a Gantt chart, this view has the program time in seconds on the x-axis and the Coven Threads on the y-axis. The colored blocks indicate various modules used in the program while the lines connecting these blocks indicate the flow of a TPH through the modules. The Coven Threads running on the same processor are grouped together so that the flow of TPHs between them can be easily seen.

Module Runtime View (Figures 5 and 7) This visualization displays the runtime of each module in a bar graph. The height of each bar represents the total amount of runtime of each module and is useful

in comparing modules. The bar graph is striped, indicating module runtime on an individual Coven Thread. It is especially helpful in determining which modules consume the most time and, therefore, should be focused on for potential optimization.

Memory View (Figure 8) Memory usage of each Coven Thread is depicted in this View. The program time in seconds, is indicated on the x-axis and the memory consumed in megabytes, is indicated on the y-axis. The graph contains different colored lines each corresponding to a Coven Thread. As modules running in each Coven Thread allocate and free memory, so does its corresponding line rise and fall in this View.

Processor Load View (Figures 9 and 10) The CPU load of each module during the Coven program execution is depicted in this View. Similar to the TPH Lifetime View, the Processor Load View depicts modules as blocks and groups Coven Threads together by processor. Each of the blocks is filled vertically in proportion to the CPU load that instance of the module placed on the processor.

5. Case Studies

In this section we look at a number of sample applications written by Coven users and how the profiler was used to identify performance problems. These examples illustrate the benefits of graphically displaying performance statistics and how end-users were able to adapt their applications to improve the performance.

These studies were conducted on a 16 node Beowulf cluster consisting of nodes with one 1GHz Pentium III processor, 1GB of RAM, and connected by Fast Ethernet. Each node of the cluster ran Red Hat Linux 8.0, Linux kernel 2.4.20, and MPICH2 0.93. All of the programs were compiled with GCC version 3.2 using maximum optimizations.

5.1. Poorly Structured Communication

Many times it is difficult to determine the amount of time a parallel program spends doing a segment of code. This can make optimization difficult in that it is sometimes not clear where performance is being lost, and where additional performance can be gained. In this example, we look at a Coven program with poorly design network communication modules. We show how the Coven profile system was used to pinpoint the performance problem.

The Coven program being examined is an image processing application that analyzes a series of images in parallel. Regions of the same image are distributed to the processors. Processors exchange boundary regions

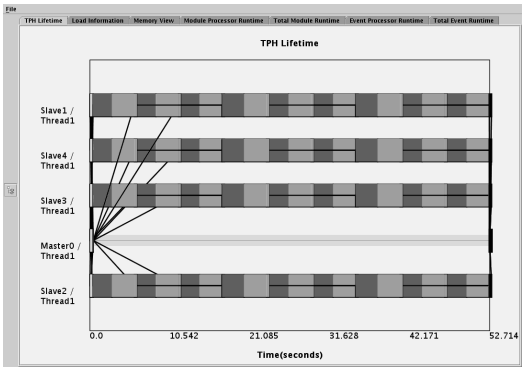

Figure 4. Timing Chart Before Optimization

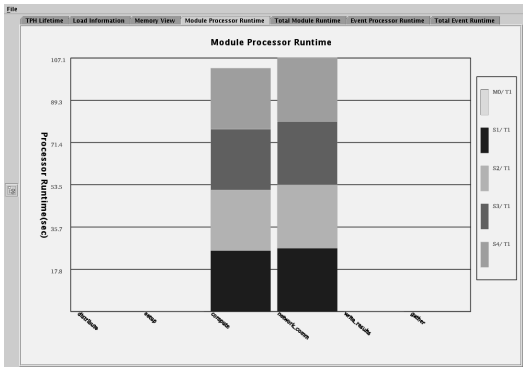

Figure 5. Module Runtime Before Optimization

with those processors that control neighboring image segments. The application requires the use of a number of intermediate MPI concepts, such as user defined datatypes and data distribution patterns. To process each segment, a computation module followed by a module which exchanges information with neighboring regions is repeated.

After executing the Coven program, we examined the performance using the Coven profiler. Figure 4 is a screenshot for a four processor version of the program. The x-axis depicts time as it progresses from program start to program completion. Each processor is depicted on a separate horizontal line. In Figure 4, the darker module is the computation module and the lighter one is the communication module. Notice that each module takes approximately the same amount of time. Figure 5 is a bargraph view of the relationship between these two modules.

Though complex, the communication phase should not be consuming as much time as the computation phase. The Coven profiler was able to reveal this information and present it in an easy to understand way. Focus was placed on improving the communication module. Figures 6 and 7 show the resulting profiler views after the communication module was optimized. Notice the total runtime of all the network modules dropped from 107 seconds to 28 seconds.

This example illustrates the Coven profiler's ability to reveal a module that is performing in a way unlike the designer intended. With this information, the Coven application developer is able to tune the program and achieve a faster runtime.

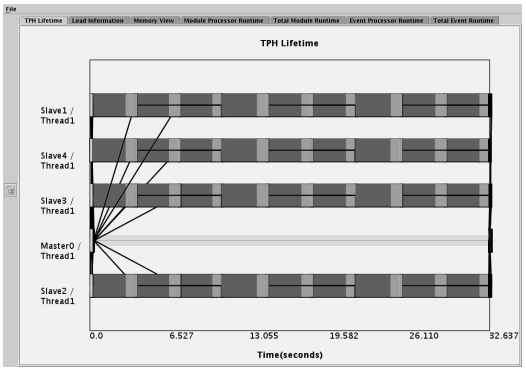

Figure 6. Timing Chart After Optimization

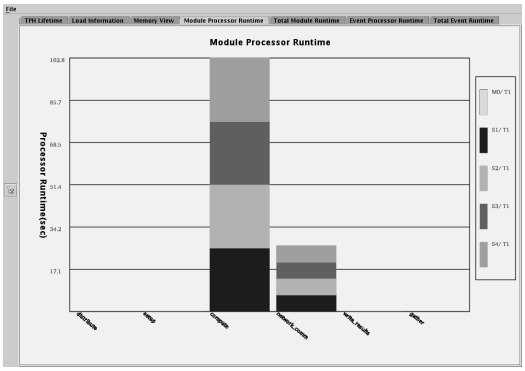

Figure 7. Module Runtime After Optimization

5.2. Memory Leak

Memory leaks can be hard to detect and even tougher to locate in complex parallel applications. Coven takes a snapshot of memory usage at many points during program execution. This can be helpful in identifying these bugs and pinpointing their exact location. This example demonstrates a Coven program with a memory leak and how the profiler was used to identify and locate the problem.

In this case study we look at a parallel n-body simulation written in Coven. Bodies in space exert forces upon each other which cause them to change their three dimensional positions. This interaction is computed in timesteps. Rather than have each processor handle the same number of bodies, each maintains a changing subset of the bodies, where every few iterations the bodies are rebalanced based on their location in space.

An initial Coven implementation of this program exhibited performance problems in that the amount of time each iteration took began to slow down at a high number of iterations. A Coven profiler was used to look at the memory usage of the program and the screenshot appears in Figure 8. Program runtime is represented on the x-axis, while memory usage appears on the y-axis. The curves represent a Coven Thread's memory usage.

In Figure 8, as time progresses the amount of memory being used progressive grows. Using this view we were able to identify the point at which the memory began to grow out of control. When used in conjunction with another view, it was easy to identify the offending module. Upon fixing the error, the memory ceased to grow every few iterations and the program began to perform as expected.

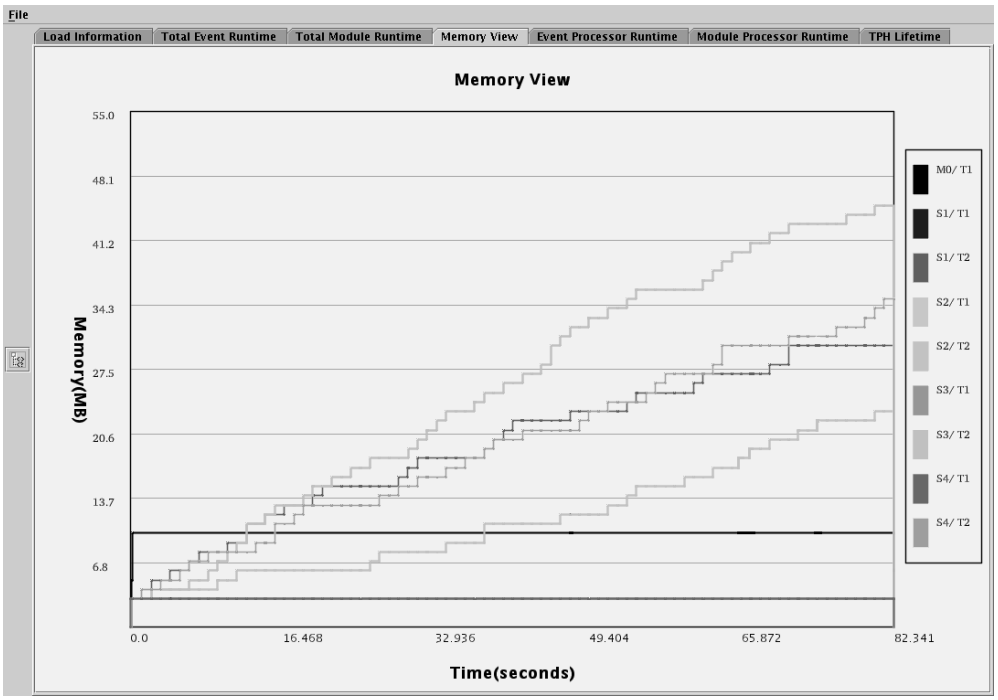

Figure 8. Memory Leak Example

5.3. Achieving Overlapping Computation and Communication By Balancing CPU Load

Many parallel applications have segments which can be executed concurrently. Concurrency can be achieved by placing these segments in separate threads, however implementing multithreading is often a daunting task for an application developer. Multithreading is handled transparently by Coven without placing an additional burden on the application writer. Modules are organized into Coven Threads running concurrently on the same processor. This case study demonstrates a Coven parallel application which can achieve a great performance improvement through multithreading.

The final application tested was a 2D Fast Fourier Transform (FFT). The 2D-FFT is used in many applications and is often considered representative of workloads that operate on matrices that are distributed across the nodes of a parallel machine. For this application, an existing FFT library was used and calls to that library were placed into Coven modules. The FFT library chosen was The Fastest Fourier Transform in the West (FFTW)[8]. FFTW was developed by MIT and is considered one of the fastest FFT implementations available.

The 2D FFT is composed of four steps: a compute phase, a communicate phase, a compute phase, and another communicate phase. The Coven program was constructed in a series of modules and several 2D-FFTs were to be processed. Each FFT was partitioned among eight processors and processed in parallel. Initially, a single Coven

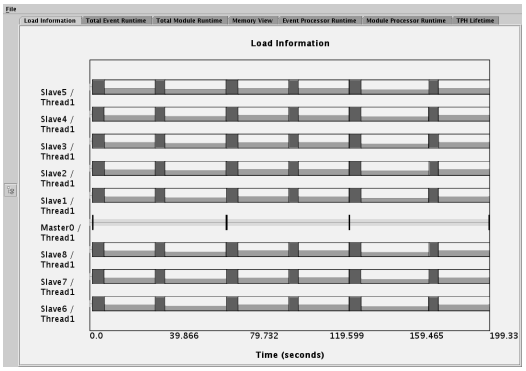

Figure 9. FFTW Single Threaded

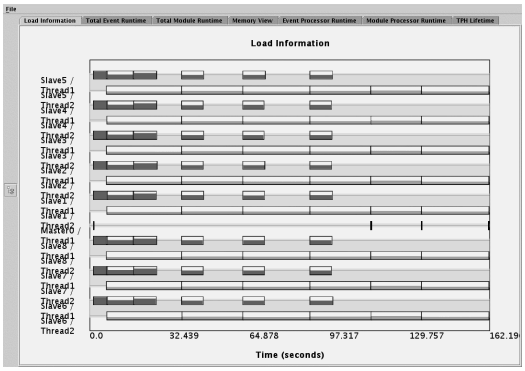

Figure 10. FFTW Multithreaded

Thread was used and the CPU Load Information view of the Coven profiler for this application can be seen in Figure 9.

In this figure, there are eight slave processors represented as horizontal lines. Each module is represented by a rectangle where the horizontal width is the time it took for the module to execute. In this application, the short modules are the computation phases while the modules which took about three times as long each are the communication modules. As is well know, the 2D-FFT is communication bound, particularly on cluster using an off-the-shelf network.

The rectangles in Figure 9 not only depict the amount of time modules executed, and in what order, but also represent the load characteristics of each module. The vertical height of each rectangle represents the amount of load that that module placed on the CPU. The computation modules fully loaded the processor, and therefore are full. The communication modules, however, only loaded each processor about 30%.

This information led us to try multiple Coven threads in an attempt to allow another 2D-FFT to begin computing during the long communication phases when little CPU was being consumed. The resulting screenshot appears in Figure 10.

As seen in figure 10, the communication modules were moved to a separate thread. Notice that during the lightly loaded communication phase the computation modules were able to execute. This simple change resulted in a program speedup which led to a decrease in program runtime from about 200 seconds to 162 seconds, a 19% improvement in performance.

While this is only applicable for programs that have sections of their code which can benefit from asynchronism, the Coven profiler is powerful enough to provide programmers tools to assist them in these cases.

6. Conclusions and Future Work

Identification and tuning of performance problems in parallel applications can be difficult, even for parallel computing experts. Since problem solving environments generally target application domain scientists and engineers who are often unfamiliar with the intricacies of parallel application performance tuning, tools which assist them in these tasks are invaluable.

Coven's profiling system provides a number of visualizations that help to tune applications built within Coven. We have shown in this paper how several different applications could be tuned to overcome performance problems with the use of the profiler.

As Coven continues to evolve and provide more features, the Coven profiler will evolve as well, and facilitate application performance tuning. Load balancing information and memory page fault statistics are being studied. In the future, the Coven profiler will not only displays this information in a helpful manner, but also suggest optimizations.

7. Acknowledgments

This work was supported by the ERC Program of the National Science Foundation under Award Number EEC-9731680. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

References

[1] Anthony Chan and William Gropp and Ewing Lusk. Performance Visualization for Parallel Programs. http://www-unix.mcs.anl.gov/perfvis/download/index.htm.

[2] Anthony Chan and William Gropp and Ewing Lusk. User's Guide for MPE: Extensions for MPI Programs. http://www-unix.mcs.anl.gov/mpi/mpich/docs/mpeman/mpeman.htm.

[3] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a Common Component Architecture for High-Performance Scientific Computing. In *Proceedings of the 1999 Conference on High Performance Distributed Computing*, pages 115–124, 1999.

[4] J. de St. Germain, J. McCorquodale, S. Parker, and C. Johnson. Uintah: A massively parallel problem solving environment. In *Ninth IEEE International Symposium on High Performance and Distributed Computing*, pages 33–41. IEEE, Piscataway, NJ, Nov 2000.

[5] J. D. de St. Germain, A. Morris, S. G. Parker, A. D. Malony, and S. Shende. Integrating performance analysis in the uintah software development cycle. In *International Symposium on High Performance Computing (ISHPC-IV)*, pages 190–206, School of Computing, University of Utah, May 2002.

[6] N. A. DeBardeleben, W. B. Ligon III, S. Pandit, and D. C. Stanzione Jr. Coven - a framework for high performance problem solving environments. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002)*, pages 291–298, Edinburgh, Scotland, UK, July 2002. IEEE Computer Society.

[7] N. A. DeBardeleben, W. B. Ligon III, and D. C. Stanzione Jr. The Component-based Environment for Remote Sensing. In *Proceedings of the 2002 IEEE Aerospace Conference*, pages 6–2661–6–2670, March 2002.

[8] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 1381–1384, Seattle, WA, May 1998.

[9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.

[10] C. Johnson, S. Parker, D. Weinstein, and S. Heffernan. Component-based problem solving environments for large-scale scientific computing. *Journal on Concurrency and Computation: Practice and Experience*, (14):1337–1349, 2002.

[11] MCS Division, Argonne National Laboratory. MPICH2. http://www-unix.mcs.anl.gov/mpi/mpich2.

[12] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tools. *IEEE Computer*, 28:37–46, 1995.

[13] Pallas GmbH. VAMPIR: Visualization and Analysis of MPI Resources. http://www.pallas.de/pages/vampir.htm.

[14] S. Shende and A. D. Malony. Integration and application of the tau performance system in parallel java environments. In *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, pages 87–96, University of Oregon, Eugene, Oregon, June 2001.